The 12th International Symposium on Virtual Reality, Archaeology and Cultural Heritage VAST
(2011)
M. Dellepiane, F. Niccolucci, S. Pena Serna, H. Rushmeier, and L. Van Gool (Editors)

Real-time Rendering of Massive Unstructured Raw Point Clouds using Screen-space Operators

Ruggero Pintus, Enrico Gobbetti, and Marco Agus

Visual Computing Group - CRS4, Italy – http://www.crs4.it/vic/

Abstract

Nowadays, 3D acquisition devices allow us to capture the geometry of huge Cultural Heritage (CH) sites, historical buildings and urban environments. We present a scalable real-time method to render this kind of models without requiring lengthy preprocessing. The method does not make any assumptions about sampling density or availability of normal vectors for the points. On a frame-by-frame basis, our GPU accelerated renderer computes point cloud visibility, fills and filters the sparse depth map to generate a continuous surface representation of the point cloud, and provides a screen-space shading term to effectively convey shape features. The technique is applicable to all rendering pipelines capable of projecting points to the frame buffer. To deal with extremely massive models, we integrate it within a multi-resolution out-of-core real-time rendering framework with small pre-computation times. Its effectiveness is demonstrated on a series of massive unstructured real-world Cultural Heritage datasets. The small precomputation times and the low memory requirements make the method suitable for quick onsite visualizations during scan campaigns.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Picture and Image Generation—; Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—.

1. Introduction

Acquisition devices, such as mobile or aerial laser scanners, are able to produce massive datasets. The abundance of samples finds a perfect field of application in Cultural Heritage (CH), where both dense and extensive sampling is required. Terrestrial Laser Scanners (TLS) or vehicle based mobile systems (Kinematic TLS) allow us to acquire high quality models of huge CH sites, historical buildings and urban environments. Wide range scanning is increasingly important, since much of CH items revolves around complex modern and ancient cityscapes. Although the typical data representation used in CH is the triangulated mesh, recently there has been a renowned interest towards the use of point clouds, which are easier to create and manage. Given this tendency, CH applications involving point clouds are becoming more and more important [SZW09, DDGM*04]. In many cases, acquired point clouds consists in an unstructured list of positions of sampled points, without other associated geometrical attributes; for instance, in the last decade the use of unstructured point clouds from moving LIDAR devices has gained a lot of attention in archaeological survey. In addition, due to high acquisition ranges, sampling rates are widely varying, and most datasets are affected by noticeable noise. Thus, fast pre-processing and real-time high-quality rendering of such noisy unstructured point clouds is crucial, especially for quick on-site visualizations during scan campaigns.

Some real-time methods directly draw the points as constant size disks. They don't fully take into account occlusions, leaving holes in under-sampled areas, producing hard to read images, and requiring color attribute to provide information on surface shape. Other techniques require extensive pre-computations of per-point normal and spatial extent. They provide higher quality rendering, but the preprocessing is time consuming and non-trivial, especially for non-uniformly-sampled massive models. We propose a method for good quality direct rendering of unstructured raw point clouds, which exploits the high performanc of GPUs to do all the computations at rendering time, on a frame-by-frame basis. We introduce an on-the-fly visibility estimator, which transforms the point rendering problem into a simpler image infilling problem applied to frame buffer

pixels, obtaining a continuous surface representation of the sparse point cloud. We also propose a shading algorithm that uses a simple screen-space method to effectively convey shape features. Unlike current high quality point rendering approaches, we do not require point attributes, such as normals and radii, minimizing pre-computation times. To deal with massive models, we integrate our approach within a multi-resolution out-of-core real-time rendering framework with fast pre-processing. Our technique is easy to implement and provides good performance and quality in rendering of massive unstructured raw point clouds. The lower memory needs and faster construction times with respect to methods requiring sampling densities and normals, make our method suitable for a variety of CH applications.

2. Related work

Point-based graphics is an extensively studied field. Here, we are going to discuss only techniques closely related to ours. For an established survey, see Gross et al. [GP07].

Since points have no extent, the major problem in rendering is to achieve smooth interpolation of surfaces, resolving visibility. Point rendering can be approached either with object-space or screen-space methods. Object-space techniques assign a size and orientation to the point replacing it with a surface element (e.g., [PZvBG00, RL00, PSL05]). Screen space methods start from pixels in the framebuffer, and apply screen-space splatting [ZPvBG01], ray tracing [SJ00], and surface infilling techniques to eliminate holes [GD98, GBP04, KCLU07, MKC08, RL08].

Despite their high image quality and good performances, they all suffer from the limitation that they need additional point attributes, such as normals and/or influence radii, for computing point set visibility and performing surface reconstruction. Some techniques [GRDE10, KK05, CS09] do not rely on normals, but still use pre-computed estimations of point density. Often, models with attributes are not available directly from the acquisition devices (e.g., LIDAR data in LAS format). While attribute computation is simple in structured range-maps [XC04], massive datasets are typically unstructured, leading to time consuming pre-processing step to compute them. Wimmer and Scheiblauer [WS06] developed a rendering system for massive unstructured raw point-clouds with a fast pre-processing based on Sequential Point Trees [DVS03]. Without point attributes, that method cannot solve visibility when point clouds are viewed from too near a distance and it doesn't provide any surface reconstruction or shading.

Unlike previous techniques, our method renders in real-time and on a frame-by-frame basis massive unstructured raw point clouds, solving both visibility and surface reconstruction with screen-space operators and without point attributes. Direct visibility of point sets has gained interest in recent years. Apart from general analytic algorithms (e.g. [DD02]), recent methods try to explicitly solve visibility of point sets without attributes. Given the position of the camera, some approaches [KTB07, MTSM10] use the *"Hidden" Point Removal* operator that solves visibility in object-space computing the convex hull of the model. The use of this approach in a screen-space framework is not straightforward, and it is not fast enough for real-time rendering. We use instead a simple screen-space visibility estimator, and an infilling method based on bilateral filtering (see, e.g., [Por08]).

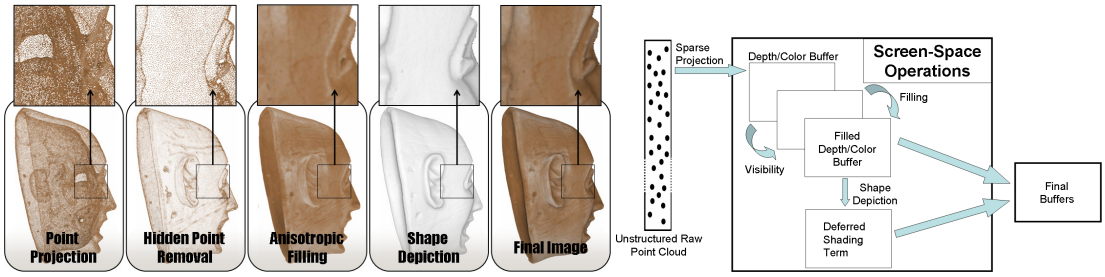
Without normal attributes, techniques must be introduced to visually convey shape information. A straightforward solution could be a screen-space normal estimation from depth buffer [KK05, KAWB09] for a directional lighting model or for a more complex lighting. It has been demonstrated (e.g., [LB99]) that a diffuse global lighting is preferable for depth discrimination. A number of GPU-based techniques have been proposed for approximating global illumination or shadow maps [SA, BSD08, VPB*09, LS10, Mit07, DL06]. Further, NPR techniques increase detail perception by drawing lines [LMLH07], toon shading [BTM06] or estimating curvature for shape depiction [VBGS08]. Using a single fast screen-space operator, our simple shape depiction method incorporates the effect of a directional lighting model, an ambient occlusion term and a non-photorealistic line drawing shape abstraction. It proves to give enough cues to understand the object curvature and to convey shape features of very noisy, non-uniformly sampled point clouds.

3. Technique overview

The proposed technique is a multi-pass GPU-based rendering pipeline implemented using vertex and fragment shaders. Fig. 1(a) and Fig. 1(b) provide an overview of how our pipeline can render high-quality images starting from an unstructured point cloud. First the point set is projected into a texture. The visibility pass decides whether each pixel is visible, providing a valid information for the current point of view. The filling operator fills all non-visible values performing a multi-pass surface reconstruction from a two-dimensional sparse depth and color buffers of visible points based on an edge-preserving image infilling. We obtain an interpolated 2.5D geometry and color. The last off-screen step consists in a multi-pass approach to compute a deferred shading term that highlights surface curvature, features and the overall shape of the rendered model. We combine all the computed textures (filled 2.5D depth and color, and shading) to produce the final image of the scene.

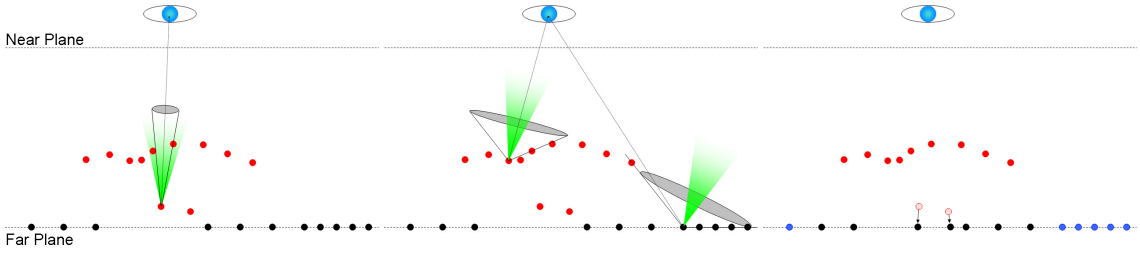
4. Visibility

Given a point set and the position and orientation of a camera, the visibility operator computes which points lie on the object surface visible from that camera. The proposed approach is based on a simple insight: if there are some points



(a) Pipeline overview (b) Pipeline structure

Figure 1: ***Overview of the rendering pipeline.*** *We first render the geometry into a texture. Then we compute point set visibility in screen space, removing invisible points. After the depth values of invisible points are filled and a screen-space multi-level shading term is calculated in order to help conveying object geometry and its fine details. Finally the filled color signal is multiplied by the shading factor.*



(a) Occluded point (b) Visible projected and background point (c) Visibility output

Figure 2: ***2D Visibility.*** *We trace a ray point-to-camera and we build the largest aperture cone that doesn't contain any other point. A small aperture cone defines a point invisible (a). A visible point has a big aperture (b). Invisible points (black dots) are set equal to 1 in depth and are mutable (in next pipeline steps), while visible points (red and blue dots) are immutable (c).*

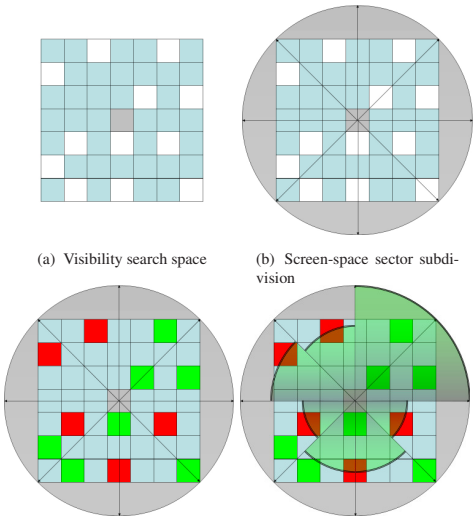
that reduce the "accessibility" of a point with respect to the camera, this point is marked as occluded; in other words, a point is visible if it is possible to build a large enough solid angle viewed by the point that contains the point-to-camera line and has no other points inside. Further, the purpose of visibility operator is also to give some constraints that could better drive the anisotropic filling operator, which cannot rely on points attributes (e.g., influence radii) to avoid boundary artifacts in the surface reconstruction task.

Straightforward screen space visibility is computed at pixel level simply enabling the depth test culling during the geometry rendering pass. It obviously leaves a lot of false-visible pixels in the frame buffer. Thus, for each pixel we need to check if it is occluded by neighboring points in the depth buffer. For simplicity we show the algorithm in 2D (Fig. 2). Red dots are projected points that has passed depth test culling, while black dots are background pixels. We trace a ray between each point and the camera, and find the maximum angle that contains this ray, has the point as apex and have no other points inside. A threshold on this angle define a point visible or not. Fig. 2(a) and Fig. 2(b) respectively show an occluded and a visible projected point. The green angle is the visibility angle, i.e., minimum angle that defines a point visible. We apply our operator to background pixels too (Fig. 2(b)), to prevent the anisotropic filling from expanding boundary and creating reconstruction artifacts; pixels marked as visible remain unchanged over the next pipeline step. In Fig. 2(c) red and blue dots are visible pixels, while black dots are marked invisible.

In 3D, we define an object space solid angle as visibility threshold. Given a pixel in the depth buffer (grey pixel in Fig. 3(a)) we compute the maximum solid angle viewed by the corresponding 3D point. Given the discrete nature of the screen-space domain and since points have no dimension, we subdivide the region around the pixel in sectors (Fig. 3(b)) and, for each of them, we compute the largest cone sector that has the point-to-camera ray as a generatrix line and doesn't contain any other point. Each cone sector defines a solid angle and if the integral of all angles from all cone sectors is bigger than the threshold, the point is marked as visible. We compute a normalized vector $\vec{v}$ that goes from the 3D position p_0 of the center pixel to the camera. We take a sector and, for each neighbor p_i within, we take another normalized vector $\vec{c} = \frac{p_i - p_0}{\|p_i - p_0\|}$ and we compute $\vec{v} \cdot \vec{c}$. The biggest dot product represents an horizon pixel that defines the maximum cone aperture for the chosen sector. Fig. 3(c) and Fig. 3(d) respectively show the horizon pixels (red) and resulting cones for each sector.

(a) Projected points (b) Visible points (c) Anisotropic filling (d) Ground truth (e) Depth buffer error

Figure 4: ***Surface reconstruction error.*** *We consider the projected point cloud of a Buddha model (a) and we apply visibility (b) and anisotropic filling (c) operator. We compare (c) with the ground truth depth buffer (d) from the original triangle mesh and we obtain the error (e). The error is normalized to the 1% of the maximum screen-space depth error.*



(a) Visibility search space (b) Screen-space sector subdivision

(c) Horizon pixels (d) Maximum cones

Figure 3: ***3D Visibility.*** *For a pixel (grey pixel) we consider a neighborhood of projected points (white pixels) (a). We subdivide this region in sectors (b). We find horizon pixels (c) and we compute the solid angle corresponding with the maximum cone (d). If the integral of cone solid angles is bigger than a fixed threshold, the point is marked as visible.*

5. Anisotropic Filling

The visibility pass produces a depth (and color) frame buffer with two kinds of pixels: already filled (visible) and empty. The value is unknown for empty pixels, while it is fixed for visible ones. It turns a 3D problem of surface reconstruction into the 2D problem of finding a meaningful value for unknown pixels starting from a sparse two-dimensional field. For solving this problem, we choose an iterative anisotropic filtering approach implemented in a multi-pass GPU fragment shader.

Throughout all passes, we use a 3x3 pixel neighborhood and we apply the operator only to empty pixels. In the first pass, all empty pixels have a non-valid depth (color) value. At each step, for each pixel we perform two possible operations. If it is non-valid, we initialize it to the median of valid depth values in its neighborhood (if any), otherwise we leave it unchanged. The choice of median instead of average comes from an edge preserving filling strategy. If the depth is valid, we update its depth and color with a weighted sum of the neighbors. Each of them has a weight equal to

$$w_i = \left(1 - \frac{r_i}{2}\right)\left[1 - min\left(1, \frac{|z_i - z_0|}{\delta}\right)\right] \qquad (1)$$

where r_i is the distance in pixel from the center, while z_i and z_0 are respectively the depth value of the neighbor and the current empty pixel. $\delta > 0$ modulates the contribution of depth distance, and $\frac{r_i}{2}$ ensures that the radial weight is non-zero for pixels inside the kernel and is zero for the closest pixel outside the kernel. The number of iterations should be twice the visibility kernel size in pixel, that implicitly define the biggest possible hole to fill. The output consists in two filled textures (depth and color). Fig. 4 compares the reconstructed depth buffer of a Buddha point cloud with the ground truth depth buffer obtained by the corresponding triangle mesh. Fig. 4(c) and Fig. 4(d) prove how our proposed method reliably approximates the real surface from its sparse representation. Fig. 4(e) shows the absolute value of the difference between reconstructed and ground truth surface; the minimum value is zero (black pixels), while the white pixels correspond with an error higher than 0.5% of the screen-space depth. As for other methods for screen-space surface reconstruction from sparse data the error is concentrated along object contours.

6. Shape depiction

After the filling step we have a complete 2.5D surface (depth and color). However, relying on color signal is not enough for the user to properly convey object geometry and features from a single view; moreover, sometimes color attribute is missing. We need to perform some deferred shading that helps to effectively convey surface shape. One solution is to compute screen-space normals and to apply a standard deferred shading method. Moreover, our interest is not a photo-realism effect, but a fast way to visualize object details for an enhanced model readability. We propose a multi-level approach that, without explicitly computing normals from depth buffer, incorporates in a single, simple and fast screen-space operator three types of shading contribution: directional light, ambient occlusion and line drawing shape abstraction.

We start from the filled depth map D^0; we consider a 3x3 neighborhood and compute, for each pixel i, the expected depth value μ^0 and its variance σ^0. To remove outliers, we define a depth interval $\left[d^0_{i_{min}}, d^0_{i_{max}}\right]$, where $d^0_{i_{min}} = \mu^0 - \sigma^0$ and $d^0_{i_{max}} = \mu^0 + \sigma^0$. Our shading term at level $k = 0$ (Fig. 5(a)) is obtained by the following equation

$$\omega^0_i = \underset{[0...1]}{clamp}\left(1 - \frac{\left|d^0_i - d^0_{i_{min}}\right|}{d^0_{i_{max}} - d^0_{i_{min}} + \varepsilon}\right) \tag{2}$$

where d^0_i is the depth of the pixel i and ε is a non-negative term, very small in a numerical sense, that avoids having a zero denominator. This term behaves as a directional light shading, because, if the surface is a plane facing the viewer, then $d^0_{i_{min}} = d^0_{i_{max}} = d^0_i$ and $\omega^0_i = 1$, while it decreases as the plane normal becomes orthogonal to the point-to-camera direction. This effect is evident in the smooth parts of David's face (Fig. 5(a)) or in the floor in front of the church entrance in Fig. 7(b). Due to a conservative approach for outlier removal that results in an interval defined by mean and variance instead of minimum and maximum depth values, ω^0_i saturates in the presence of sharp edges producing dark or white lines over the surface. The line drawing effect is visible in David's hair (Fig. 5(a)) or in the sharp contours of buildings in Fig. 8(b).

(a) Level $k = 0$ (b) Level $k = 10$ (c) Level $k = 30$

Figure 5: ***Our shading term at different smoothing levels.*** *Our shading term computed at each smoothing level; (a) base level, (b) after 10 , (c) after 30 iterations.*

Similarly to Rusinkiewicz et al [RBD06], we introduce a multi-level computation to add a global lighting term to our deferred shading. We consider D^k as the depth signal after k smoothing steps, and at each step we set the $(k+1)$-depth as the average of the kth mean value and the kth lower depth limit $d^{k+1}_i = \left(\mu^k + d^k_{i_{min}}\right)/2$, reducing the number of necessary smoothing steps in a logarithmic manner. For each depth smoothing level we found the corresponding ω^k_i. Fig. 5(b) and 5(c) show this term for two of these levels.

At the end of K multi-level off-screen rendering passes, we merge all ω^k_i in a single shading signal as

$$\omega_i = \sum_{k=0}^{K-1} \frac{\omega^k_i}{(k+1)^\xi} \tag{3}$$

where ξ decides how the weight decreases as a function of smoothing level. The more is ξ, the sharper is the shading. The less is ξ, the more is the contribution of the global lighting term to the final deferred shading. Fig. 6 shows the shading with a constant weight (i.e. $\xi = 0$) (Fig. 6(a)), with $\xi = 1$ (Fig. 6(b)) and with $\xi = \frac{3}{2}$ (Fig. 6(c)). We choose this latter weight for the experiments in Sec. 8. Finally, Fig. 7 compares a general screen-space global illumination term with our hybrid shading.

(a) Volumetric Obscurance (b) Our shading

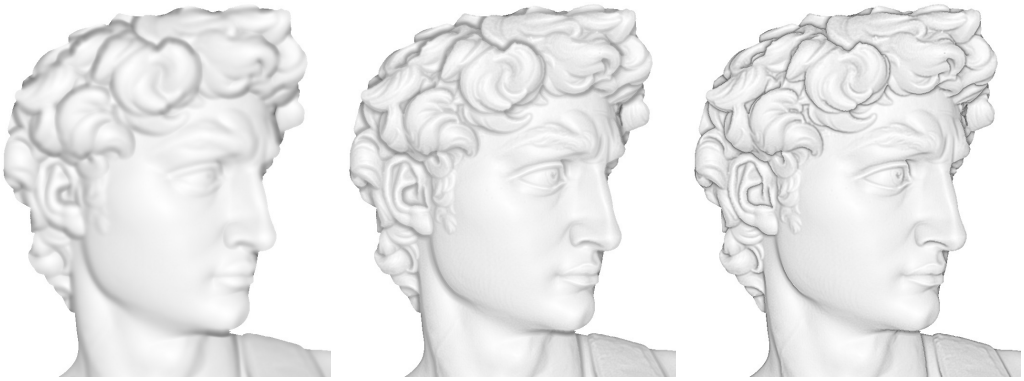
Figure 7: ***Volumetric obscurance vs. our shading.*** *A comparison between a screen-space obscurance algorithm [LS10] (a), and our term (b), which is a mix of directional light, ambient occlusion and line drawing shading.*

7. Multi-Resolution structure

The proposed technique was integrated in a multi-resolution framework. We employed a data representation based on a multi-resolution point hierarchy, to randomly access different portions of the dataset to compute view-dependent cuts of the graph during real-time rendering. The input is a set of unstructured N positions, and eventually colors. Our multi-resolution structure is a coarse-grained kd-tree partition of the input dataset, with leaves containing less than a predefined target sample count and with inner nodes containing a filtered version of their children, with a number of samples equal to the target count. The multi-resolution structure is built off-line. First, we start adding points into an external memory array. If the number of points N is less than a fixed sample count M, a leaf node is generated , otherwise samples are distributed to the two children by bisecting the bounding box at the midpoint of its longest axis, and recursively continuing the partitioning procedure. Then, we build inner nodes from their children: the parent node is subdivided in a fixed-size grid of M cells and each cell conatins one sample created by merging children samples.

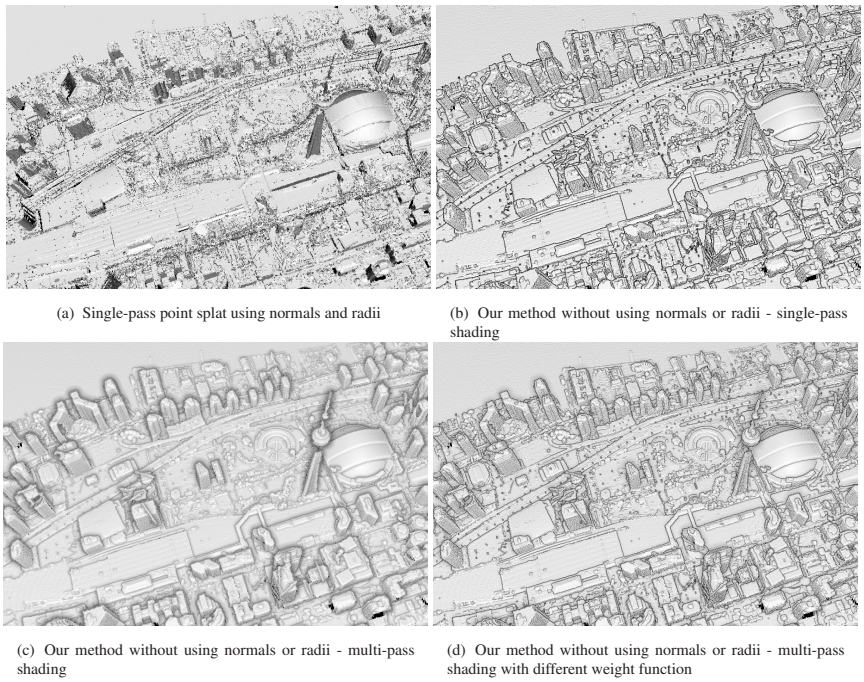
8. Results

Our method was implemented on Linux using C++ with OpenGL. In Table 1 we list the models used for testing: *David* and *Ostiano* are acquired using triangulation and time-of-flight scanner, while *Sirmione*, *Loggia* and *Toronto* are noisy unprocessed raw LIDAR datasets. First we build an out-of-core multiresolution hierarchy. Our performances, computed with a processor Intel 2.4GHz, are comparable



(a) Level weight = *const* (b) Level weight $\propto (k+1)^{-1}$ (c) Level weight $\propto (k+1)^{-\frac{3}{2}}$

Figure 6: ***Level contribution for final deferred shading.*** *Different types of weights for the kth level contribution in the final deferred shading computation: (a) constant contribution, (b) and (c) two different hyperbolic contributions.*



(a) Single-pass point splat using normals and radii (b) Our method without using normals or radii - single-pass shading

(c) Our method without using normals or radii - multi-pass shading (d) Our method without using normals or radii - multi-pass shading with different weight function

Figure 8: ***Toronto (260Mpts):*** *(a) single pass point splat using normals and radii, and our method with (b) single-pass or (c) multi-pass shading. (d) multi-pass shading with increased weight in the first pass. With such noisy data our method proves to be very effective and it helps to appreciate the city layout and gives enough cues to better understand the depth information.*

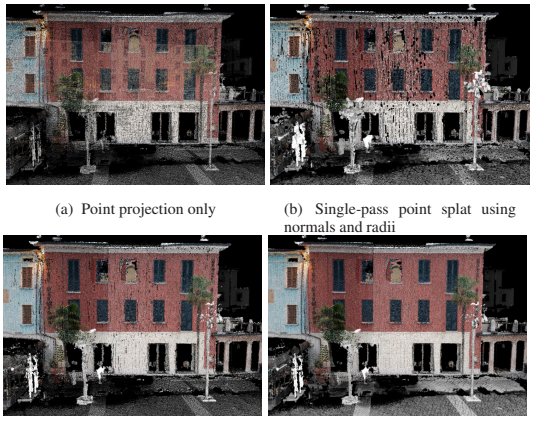
with state-of-the-art fast pre-processing methods for massive unstructured raw point cloud rendering. Wimmer et al. [WS06] built a 260Mpts model and no attribute computation in 2 hours with a processor Intel 3.2GHz, and Wand et al. [WBB*07] took 100 mins for a 76Mpts dataset. Our point pre-processing rates (without attribute computation) range from $30K points/sec$ to $60K points/sec$.

The algorithm was tested on an Intel 3.2GHz and a NVidia GeForce GTX 480. Depending on the sparse nature of the dataset, to obtain the proper rendering quality, the user has to choose the visibility kernel. The user could choose a single pass shading, that results in a sharp visualization with directional lighting and line drawing effects, or a multi-pass term to include a global illumination too. These signals can

Model	Points	Model	Points
David	470Mpts	Ostiano	213Mpts
Sirmione	100Mpts	Toronto	260Mpts
Loggia	110Mpts		

Table 1: ***Dataset sizes.***

be optionally merged with different weight functions to have the desired visualization. Here we use visibility kernel sizes ranging from 4x4 to 9x9 pixels. Further various shading conditions are considered. For a 1024x768 window size, we experimented a frame rate going from 15fps up.



(a) Point projection only (b) Single-pass point splat using normals and radii

(c) Pull-push using normals and radii (d) Our method without using normals and radii

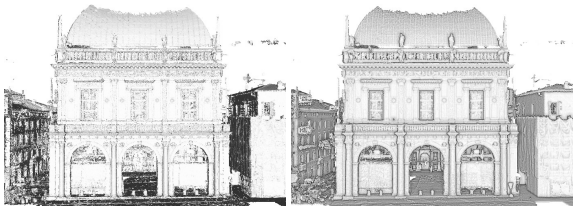
Figure 9: ***Sirmione (100Mpts):*** *(a) point projection without visibility estimation, (b) single-pass point splat and (c) pull-push infilling with normals and radii, and (d) our visibility and infilling without normals, radii and shape depiction.*

The first data is the city of Toronto, a complex urban environment. It is very noisy and our method proves to be very effective in its visualization. Fig. 8(a) shows the single-pass point splat rendering, while Fig. 8(b) and Fig. 8(c) are two renderings with our method using respectively a single pass and a multi-pass shading approach. A different weighting function results in the shading effect of Fig. 8(d); here we assign to the first shading pass a weight equal to the sum of the weights of all other shading steps. This signal helps to appreciate the city layout and gives enough cues to better understand the depth information too. Another dataset is the city of Sirmione (Fig. 9 and Fig. 10). Fig. 9(a) shows the straightforward visualization of the projected points in a view close to a building. The model is difficult to understand because we see both the building and the rest of the data beside it. The dataset becomes clear after applying visibility estimation and surface reconstruction. Fig. 9(b) and Fig. 9(c) respectively show the results of the single-pass point splat and pull-push technique (without shading). If the data is noisy, point splat rendering gives a less clear visualization, while pull-push techniques produce better renderings; however, they use normals and radii to reconstruct the surface and implicitly solve visibility, requiring extensive pre-processing. Fig. 9(d) shows our rendering technique after the visibility and filling steps. The quality of our reconstruction is comparable to the pull-push technique, but we do not need any pre-computed attribute. In Fig. 10 we compare again our rendering (Fig. 10(c)) with a classical single-pass point splat (Fig. 10(a)) or pull-push (Fig. 10(b)) technique. Fig. 10(c) shows our multi-pass shape depiction shading technique, which helps to better convey some details and depth discontinuities that are not evident using only the visible filled model (Fig. 10(c)). Finally, in Fig. 12, we compare single-pass point splatting with our method, applied to the Loggia dataset (Brescia, Italy), while Fig. 11 shows a visualization of the church *La Pieve di Ostiano* in Cremona (Italy) with the proposed technique. Our method is more effective to perceive geometrical and color details in the presence of very noisy datasets.

(a)

(b) A detail

Figure 11: ***Ostiano (213Mpts):*** *(a) and (b) our method without using normals, radii and shape depiction. The detail proves how our edge-preserving infilling helps to perceive fine color details on a such noisy dataset.*

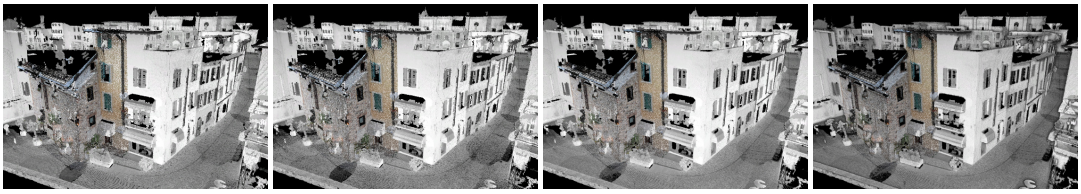


(a) Single-pass point splat using normals and radii - with shading (b) Our method without using normals and radii - with shape depiction

Figure 12: ***Loggia (110Mpts):*** *(a) single-pass point splat with shading and (b) our method with shape depiction.*

9. Conclusions and Future Works

We have presented a real-time rendering technique to visualize unstructured raw point clouds using screen-space operators and without any geometric attribute pre-computation. Our results prove the applicability to very noisy datasets. In



(a) Single-pass point splat using normals and radii

(b) Pull-push using normals and radii

(c) Our method without using normals and radii

(d) Our method without using normals and radii - with shape depiction

Figure 10: ***Sirmione (100Mpts):*** *(a) single-pass point splat and (b) pull-push infilling algorithm using normals and radii, our method without using normals and radii, without (c) and with (d) our shape depiction shading approach.*

CH, fast pre-computation makes our approach suitable for on-site quick previews, which are very important for fast visual checks during scanning campaigns. We plan to investigate other shape depiction strategies to produce more expressive visualizations.

Acknowledgments. This research is partially supported by EU FP7 grant 242341 (INDIGO). David dataset courtesy of Digital Michelangelo project. Data of Sirmione, Ostiano and Loggia courtesy of Gexcel.

References

[BSD08] BAVOIL L., SAINZ M., DIMITROV R.: Image-space horizon-based ambient occlusion. In *SIGGRAPH* (2008). 2

[BTM06] BARLA P., THOLLOT J., MARKOSIAN L.: X-toon: An extended toon shader. In *Proc. NPAR* (2006), ACM. 2

[CS09] CORDS H., STAADT O.: Interactive screen-space surface rendering of dynamic particle clouds. *JGT 14*, 3 (2009), 1–19. 2

[DD02] DUGUET F., DRETTAKIS G.: Robust epsilon visibility. *ACM Transactions on Graphics 21*, 3 (July 2002), 567–575. 2

[DDGM*04] DUGUET F., DRETTAKIS G., GIRARDEAU-MONTAUT D., MARTINEZ J.-L., SCHMITT F.: A point-based approach for capture, display and illustration of very complex archeological artefacts. In *VAST 2004* (Dec. 2004), pp. 105–114. 1

[DL06] DONNELLY W., LAURITZEN A.: Variance shadow maps. In *I3D* (2006), pp. 161–165. 2

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential point trees. *ACM ToG 22*, 3 (2003), 657–662. 2

[GBP04] GUENNEBAUD G., BARTHE L., PAULIN M.: Deferred splatting. *Computer Graphics Forum 23*, 3 (2004), 653–660. 2

[GD98] GROSSMAN J. P., DALLY W. J.: Point sample rendering. In *EGWR* (1998), pp. 181–192. 2

[GP07] GROSS M., PFISTER H.: *Point-Based Graphics*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. 2

[GRDE10] GROTTEL S., REINA G., DACHSBACHER C., ERTL T.: Coherent Culling and Shading for Large Molecular Dynamics Visualization. In *EUROVIS* (2010), vol. 29, pp. 953–962. 2

[KAWB09] KLASING K., ALTHOFF D., WOLLHERR D., BUSS M.: Comparison of surface normal estimation methods for range sensing applications. In *Proc. ICRA* (2009), pp. 1977–1982. 2

[KCLU07] KOPF J., COHEN M. F., LISCHINSKI D., UYTTENDAELE M.: Joint bilateral upsampling. *ACM ToG 26*, 3 (July 2007), 96:1–96:5. 2

[KK05] KAWATA H., KANAI T.: Direct point rendering on gpu. In *ISVC* (2005), pp. 587–594. 2

[KTB07] KATZ S., TAL A., BASRI R.: Direct visibility of point sets. *ACM ToG 26*, 3 (July 2007), 24:1–24:11. 2

[LB99] LANGER M. S., BULTHOFF H. H.: *Perception of Shape From Shading on a Cloudy Day*. Tech. Rep. No. 73, 1999. 2

[LMLH07] LEE Y., MARKOSIAN L., LEE S., HUGHES J. F.: Line drawings via abstracted shading. *ACM Transactions on Graphics 26*, 3 (July 2007), 18:1–18:5. 2

[LS10] LOOS B. J., SLOAN P.-P.: Volumetric obscurance. In *Proc. I3D* (New York, NY, USA, 2010), ACM, pp. 151–156. 2, 5

[Mit07] MITTRING M.: Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 courses* (2007), SIGGRAPH '07, pp. 97–121. 2

[MKC08] MARROQUIM R., KRAUS M., CAVALCANTI P. R.: Efficient image reconstruction for point-based and line-based rendering. *Computers & Graphics 32*, 2 (Apr. 2008), 189–203. 2

[MTSM10] MEHRA R., TRIPATHI P., SHEFFER A., MITRA N. J.: Visibility of noisy point cloud data. *Computers and Graphics In Press* (2010). 2

[Por08] PORIKLI F.: Constant time O(1) bilateral filtering. In *Proc. CVPR* (jun. 2008), pp. 1 –8. 2

[PSL05] PAJAROLA R., SAINZ M., LARIO R.: Xsplat: External memory multiresolution point visualization. In *Proc. VIIP* (2005), pp. 628–633. 2

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *SIGGRAPH* (2000), pp. 335–342. 2

[RBD06] RUSINKIEWICZ S., BURNS M., DECARLO D.: Exaggerated shading for depicting shape and detail. *ACM Trans. Graph. 25*, 3 (2006), 1199–1205. 5

[RL00] RUSINKIEWICZ S., LEVOY M.: Qsplat: A multiresolution point rendering system for large meshes. In *SIGGRAPH* (2000), pp. 343–352. 2

[RL08] ROSENTHAL P., LINSEN L.: Image-space point cloud rendering. In *Proc. Computer Graphics International* (2008). 2

[SA] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on gpus. In *I3D'07*, pp. 73–80. 2

[SJ00] SCHAUFLER G., JENSEN H. W.: Ray tracing point sampled geometry. In *EGWR* (2000), pp. 319–328. 2

[SZW09] SCHEIBLAUER C., ZIMMERMANN N., WIMMER M.: Interactive domitilla catacomb exploration. In *VAST09* (Sept. 2009), Eurographics, pp. 65–72. 1

[VBGS08] VERGNE R., BARLA P., GRANIER X., SCHLICK C.: Apparent relief: a shape descriptor for stylized shading. In *Proc. NPAR* (2008), ACM, pp. 23–29. 2

[VPB*09] VERGNE R., PACANOWSKI R., BARLA P., GRANIER X., SCHLICK C.: Light warping for enhanced surface depiction. *ACM Transactions on Graphics 28*, 3 (July 2009), 25:1–25:8. 2

[WBB*07] WAND M., BERNER A., BOKELOH M., FLECK A., HOFFMANN M., JENKE P., MAIER B., STANEKER D., SCHILLING A.: Interactive editing of large point clouds. In *PBG* (2007), pp. 37–46. 6

[WS06] WIMMER M., SCHEIBLAUER C.: Instant points: Fast rendering of unprocessed point clouds. In *PBG* (2006). 2, 6

[XC04] XU H., CHEN B.: Stylized rendering of 3d scanned real world environments. In *NPAR 2004* (June 2004), pp. 25–34. 2

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *SIGGRAPH* (2001), pp. 371–378. 2